Lec.slide 1. Micro

What is Embedded system?

↳ Combination of hardware and software.

↳ Designed to Perform specific function.

↳ combination of computer hardware with the software to control, monitor, communicate to do multiple operations

ex: Washing machine, Digital Camera, DVD Player.

General Purpose system vs. Embedded system

General Purpose	Embedded.
↳ combination of generic HW & General Purpose OS for executing variety of apps.	↳ combination of special Purpose HW and Embedded OS for executing specific set of apps.
Need not to be deterministic in execution behavior	Execution behavior is deter. for certain types of embedded system.
Example: Computer, laptop	Example: DVD, Printers.

Characteristics of Embedded systems:

* single functioned.
* Cost effectiveness
* fast execution time
* Reliability.
* low lower Consumption
* efficient use of memory.
* Processing Power.

* Major application areas of Embedded System:-

1) Customer electronics (Cameras, etc)
2) Household appliances (TV, DVD Players, Washing machine)
3) Home automation & security system (fire alarms ... etc)
4) Telecom (handset, cellular telephones ... etc)
5) Healthcare (scanners, ECG, etc)
6) Banking & retail (ATM, Currency Counter, etc)

Categorization of embedded system

[1] Based on Generalization:-

a. first generation (from 4 to 8 bit microcontrollers & ex: stepper motor control units)

b. 2nd Generation (8 to 16 bit micro Controllers & ex: sensors)

c. 3rd " (16 to 32 bit & ex: DSP)

d. 4th " (Smartphones, Mobile internet devices, etc)

[2] Based in Complexity & Performance :-

a. Small scale Embedded system

↳ small Processing elements with simple Program

↳ low cost. ↳ lower Performance.

b. Medium scale ES :-

↳ slightly complex in HW & SW.

↳ low cost. ↳ Medium performance.

c. Large scale ES

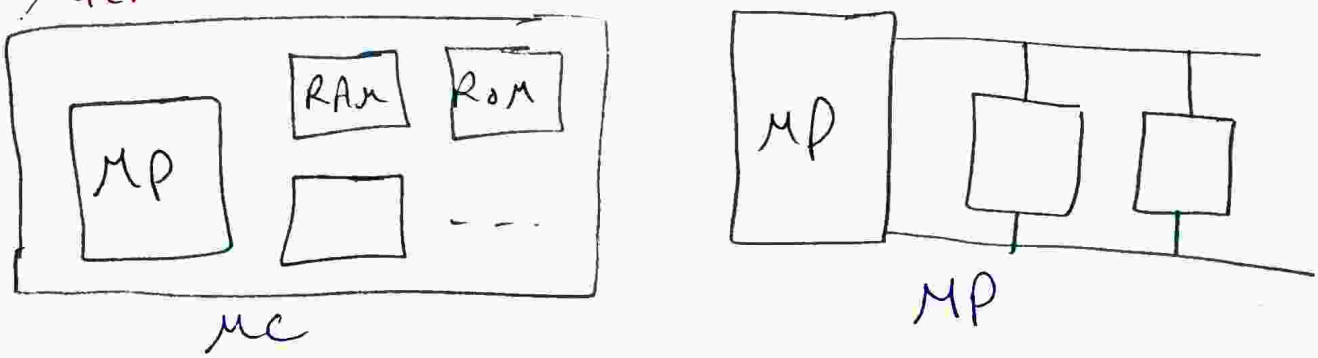
↳ complex ↳ High Performance. ↳ low cost

[3] Based on determinstic behavior.

↳ Hard real time ↳ soft real time.

[4] Based on Triggers (Event based & time based)

Micro controller vs Micro Processor



ES → uses (MC) to do specific task.

PC → uses MP to do different things.

* criteria of choosing MC

* speed * Power Consumption * Cost

* RAM & ROM * I/O Ports

* Packaging (Put MC in which Package)

* Community

Components of 8051 MC

a) 128 bytes RAM b) 4K ROM

c) 2 timers d) one serial port

e) 6 interrupt f) 4 I/O Ports (each 8 bits)

Notes

1) 8051 is subset of 8052

2) 8031 " a ROM-less 8051

	8051	8052	8031
ROM	4K	8K	0K
RAM	128	256	128
Timers	2	3	2
I/O Pins	32	32	32
Serial Port	1	1	1
Interrupt Sources	6	8	6

4

3) * Vast majority of 8051 register are 8-bit registers.

Micro Processor V-s Micro Controller

Micro Processor	Micro controller
→ use CISC Architecture	→ use RISC and Harvard.
→ has ROM, RAM, secondary storage memory, Ilo etc Placed on board and Connected through Buses.	→ ROM, RAM, Ilo Prepherals are Combined in single integrated circuit
→ less secured	→ more secured.
→ expensive	→ cheaper
→ Design takes more time	→ Design it takes less time.

RISC	CISC
simple instruction format	Variable inst. format
large set of CPU registers	small set of general Purpose registers
very few addressing modes	large number of addressing modes.
simple Pipelinging	Complex Pipelining
supports on chip cache memory	seldom supports on chip cache memory

8051 Assembly language Programming

1) How instructions are executed.

2) Registers.
   - a) 8 bit registers Me
   - b) A: Accumulator, must be destination of any operation.
   - c) B, PC, DPTR, SP
   - R_{0-7}

Any Assembly Program {instructions & directives}

instructions ⇒ [label:] opCode [operands] [; Comments]

Directives (not instructions)
- but they help assembler in assembly process.

ex ORG address ⇒ PC = address

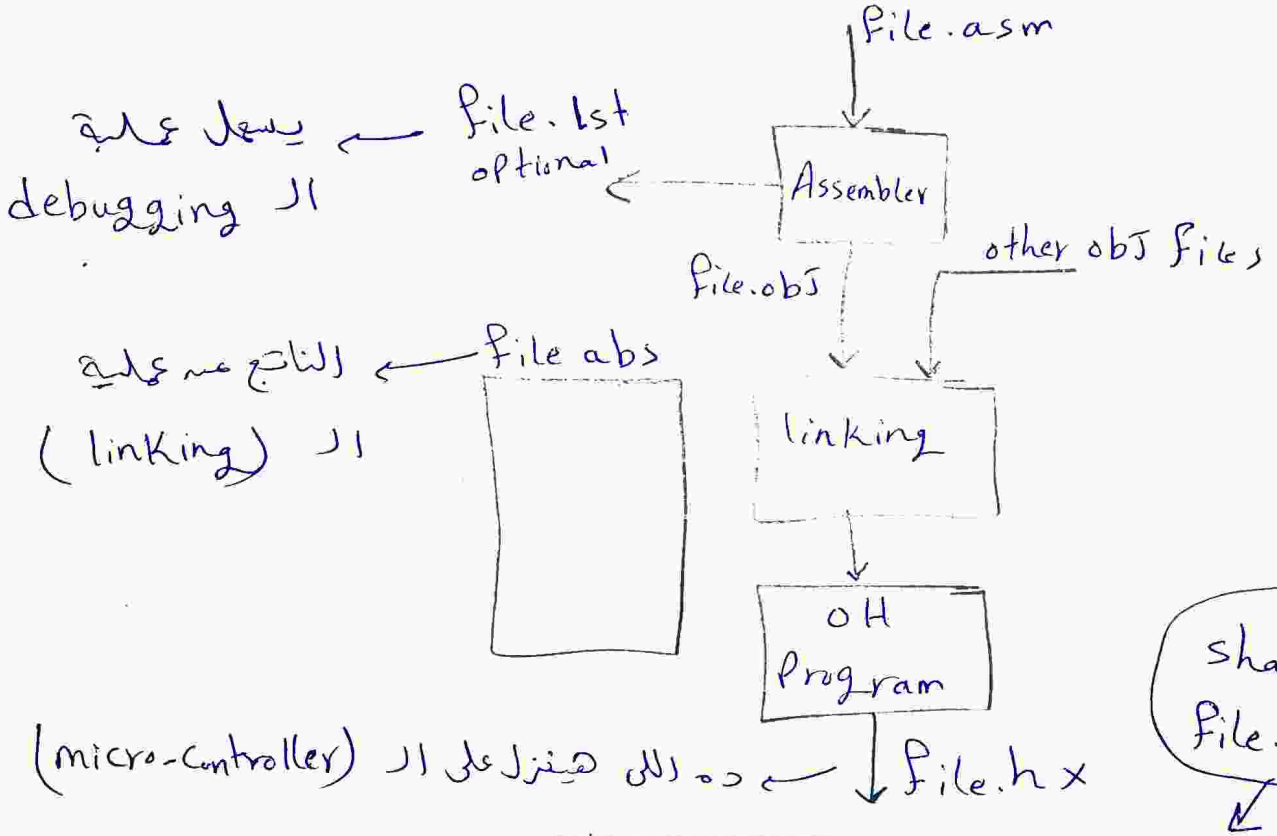
معناها إنه البرنامج بتاعي أول (instruction) هيبقى موجود في الـ (Address) اللي جنبه ده وهو اللي الـ (Program Counter) هيشاور عليه في الأول.

→ END

بيعرف الـ (Assembler) إنه دي نهاية الجزء اللي بيعمل ليه (Assembly).

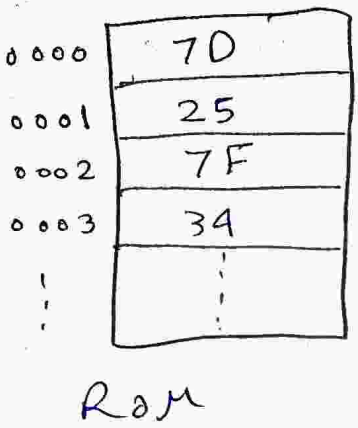
step of assembly language Program

(file.asm) بكتب فيه ال (instruction) اللى عايزها
لـ بيدخل على ال (assembler) يطلع (file.object)
و و و التانيل اللى انتا محتاج يحصل ليه (linking) مع
الحاجة اللى انتا عايز تعملها .



Address	Machine	Assembly	Comments
		ORG 0000	
0000	7D25	Mov R5, #25	
0002	7F34	Mov R6, #34	
0004	2D	ADD A, R5	
0005	—	—	

أول ما تفتح الـ (Program) هتلاقي إنه
الـ PC عندك فيه PC=0000 وكل
شوية هيعمل (increment) وهيقرا عارف
إنه الـ (instruction) اللي جاي منين



ملحوظة: 1) مقدرتش أكتب برنامج السعة بتاعته أكبر من
4K byte ← ROM لأن سعة الـ (4K byte)

2) هيديلك برنامج (assembly) ويطلب منك تحط الجدول
بتاع الـ (lst.file) في الصفحة السابقة وبعدين تعمل الجدول في
أول الصفحة دي زي مثال في ص 74، 75 في (slides)

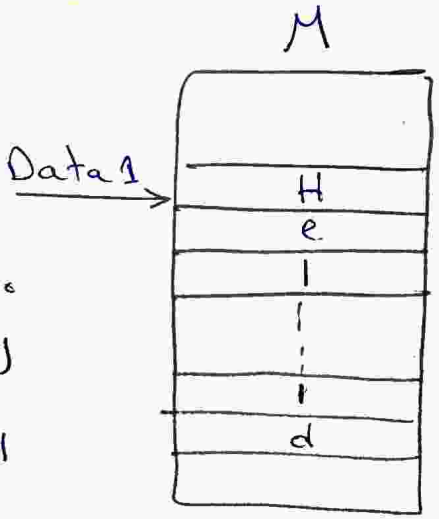
DB directive

↳ most widely used ↳ used to define 8-bit data

شرح لو عايز أطبع (Hello world) بالـ (Assembly)
لـ محتاج تكون موجودة في الـ (memory)

```
ORG 500
Data 1 DB " Hello world"
```

لما عند الـ (lable Data1) اللي موجودة عند
500 كأني بقوله إنه بيشاور على الأمر دي.
لما أنا كده عرفت بالـ (DB) مجموعة الأوامر
الموجودة عند الـ (address 500)
محجوزة لـ (Hello world)



*EQU

لما بعمل (define) لثابت مش غير مايأخد مكان في ال (memory) لما بعمل بيحط قيمة للثابت في (data label) وده لما ال (label) يظهر في البرنامج بيستخدم قيمة الثابت فوراً.

ex

COUNT EQU 25 → use EQU for Counter Constant

MOV R3, #COUNT → Constant used to load R3.

Flag register (PSW) → Program status world

لغة عبارة عن (8-bit register) بيستخدم منهم 6-bit

CY	AC	F0	RS1	RS0	OV	--	P

(not used)

Where

CY → Carry. AC → Auxiliary Carry.

RS0, RS1 → Bank selectors. OV → overflow.

P → Parity flag. F0 → user definable.

ex

```
  1 1 1
  0 0 1 1 1 0 0 0      38
  0 0 1 0 1 1 1 1    + 2F
  ---------------    ----
  0 1 1 0  0 1 1 1    67
    6        7
```

عدد الوحايد
لو odd ← P=1
P=0 ← even

CY=0 ← 0=D7 عشان

انتقل من D3 ← D4 ومعاك

AC=1 ← Carry

P=1 , AC=1, CY=0

* RAM : 128 bytes

(Variables) لـ مكان أخزن فيه الـ

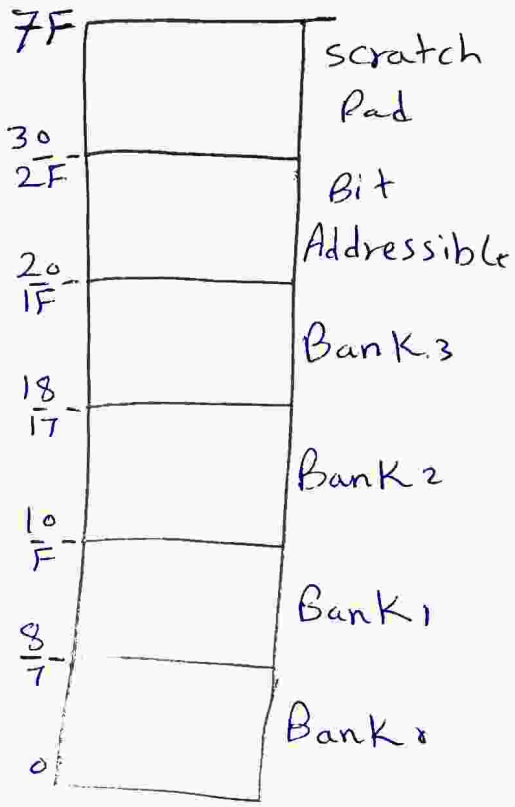
(Operations) اللي بعمل فيها

↳ (RS1, RS0)

(Bank) حسب قيمهم بختار الـ

اللي الـ (register) بتاعي بتعامل معاهم.

RS0	RS1	Bank
0	0	0
0	1	1
1	0	2
1	1	3



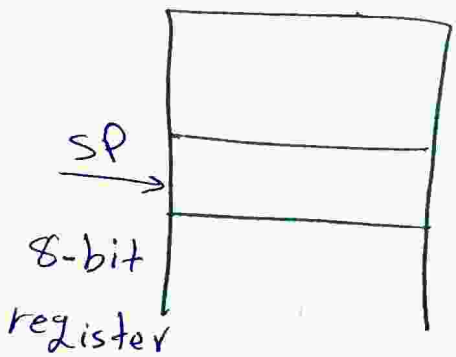
(stack) بتاعه هو (default) الـ (Bank 1)

↳ Push & Pop

Default

SP = 7

عشان (Bank 1) تبدأ من (8)



Push → storing CPU register in stack
↳ increment then Push.

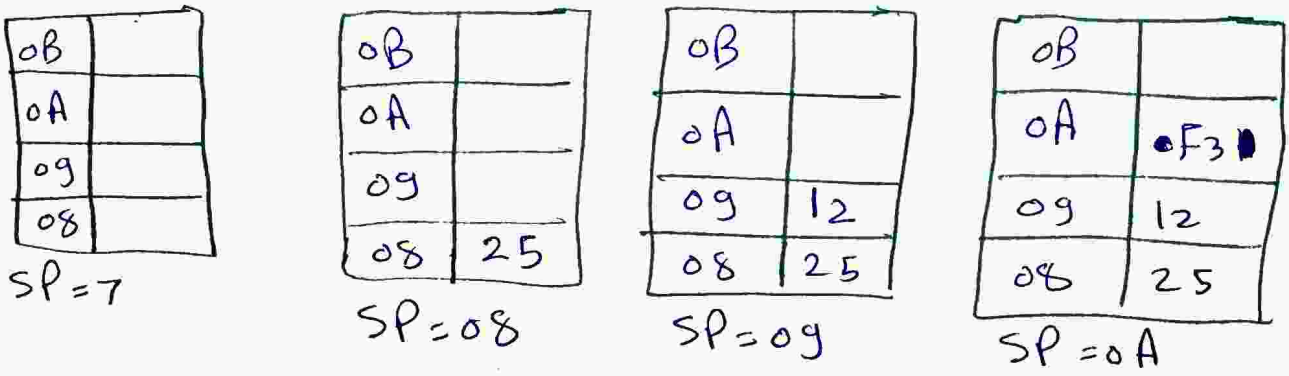
Pop → loading contents of stack back into CPU register.
↳ Pop then decrement.

ملحوظة: لا يشترط ان (SP = 7) هو فقط بيشاور على عنوان معين وبيكون العنوان اللي بعده هو اللي بتخزن فيه الـ (data) بتاعتي.

Ex show stack and stack Pointer

```
Mov R6, #25H
Mov R1, #12H
Mov R4, #0F3H
Push 6
Push 1
Push 4
```

Sol



JUMP, Loop and CALL instructions

Loop repeat sequence of instructions a certain number of times.

is performed by => DJNZ Reg, label

بعمل (decrement) لـ (reg) وبـ (check) كل شوية

لحالها ليس بوقف بعمل (Jump) لـ (label)

ex

Reg = value

— — —

— — —

DJNZ Reg, loop

بعمل (decrement) لـ (reg) وطول ماهو ليس بوقف بعمل (loop)

⇐ ماذا لو حتاج أكرر (action) أكتر من 256 مره

لأكبر قيمة لـ (reg) حيا (FF) 255 لكن لو عندي (loop) وتحتاج قيمة أكبر من (255) نعمل أكتر من (loop) جوه بعض (nested loop)

لو بقى مثلاً لو عندي 700 نعمل (2 loops) واحده 10 والأخرى 70، والناتج يكون حاصل ضرب بعض

Jump

Conditional Jump:

لـ بعمل (operation) ثم أعمل (check flag) وعلى أساسه بعرف أخذ أى (action) فى البرنامج.

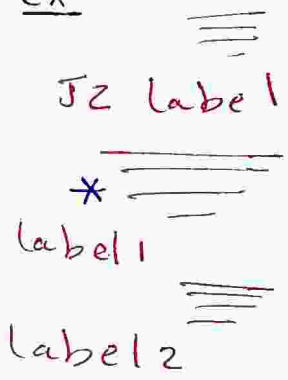
ex JZ label

لـ يعمل (check) على (zero flag) ومنها أعمل (Jump) لـ (label)

if equal to zero ⇒ Jump to label.

else (unconditional Jump) فى المنطقة * عمل (label 2) لـ

ex



Jump

1) LJMP (long Jump)

لـ بعمل (Jump) لأى مكان فى البرنامج.

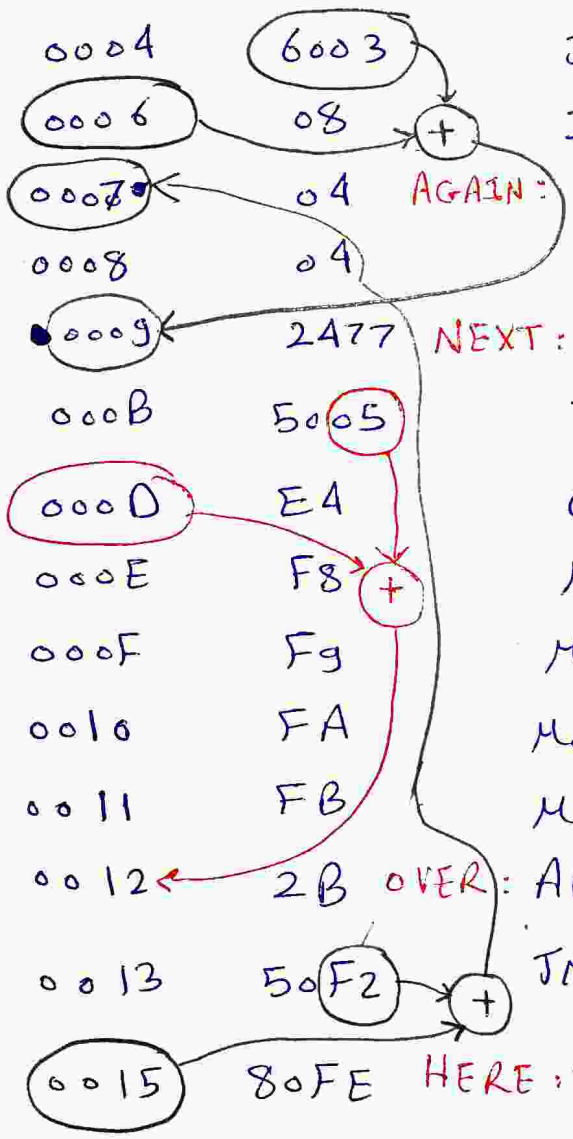
لـ لأنه بياخد (3 bytes) واحده لـ (opCode) , إتنين لـ (address)

2) SJMP (Short Jump) (-128 to 127)

لـ بقدر أعمل (Jump) فوق أو تحت بحد أقصى (128 bit) لـ بياخد (2-byte)

Page 112 ex

Line	Pc	opCode	operand
01	0000		ORG 0000
02	0000	7800	MOV R0, #0
03	0002	7455	MOV A, #55H
04	0004	6003	JZ NEXT
05	0006	08	INC R0
06	0007	04	AGAIN: INC A
07	0008	04	INC A
08	0009	2477	NEXT: Add A, #77H
09	000B	5005	JNC OVER
10	000D	E4	CLR A
11	000E	F8	MOV R0, A
12	000F	F9	MOV R1, A
13	0010	FA	MOV R2, A
14	0011	FB	MOV R3, A
15	0012	2B	OVER: ADD A, R3
16	0013	50F2	JNC AGAIN
17	0015	80FE	HERE: SJMP HERE
18	0017		END



ex

0004 60

0005 03

PC=0006 → + → 0009

هنا يجيب (opCode) لل Jump أو loop

عند JNC OVER

عايز يطلع بـ 5005

فبيطرح (0012) مكان ما الـ (OVER) راحت سه الـ (0D) الـ PC

كذلك مع JNC AGAIN

لو الـ (opCode) مكانه سه رقمين الـ PC هيزيد بـ (2)

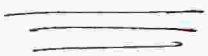
ولو رقم واحد الـ (PC) هيزيد (1)

Call

- يكتب نوبة أكواد وفي النص عايز أعمل (subroutine) فبعمل ليه (Call)

ex

Call label1



sub1:

RET return

- الـ (Call) بيعمل Push لـ (Pc) ويعمل تغيير ليه لـ (sub 1) عشان يشاور عليه

Push PC

change Pc to sub 1

sub1 :

RET ⇒ Pop PC

- قبل الجزء بتاع (Sub 1) حطوا (HLT) معناها نهاية البرنامج

Call

1) Long call (LCall)

- لما بعمل (Call) في أي مساحة في البرنامج (3-byte instruction) يعتبر
- الـ (subroutine) موجود في أي مكان خلال الـ (64K byte).

2) ACall (absolute call)

- لما بعمل (Call) في حدود (2K-byte) فقط (2-byte instruction) يعتبر

Notes

لو في ال (main) وال (subroutine) بتستعمل نفس ال (registers) ⇐ ال (call) مش معنى بأنه يعمل ال Push لأنه بيتعامل مع ال (return ، Push) فقط.

لو في هذه الحالة قبل بداية ال (subroutine) هعمل Push لقيم ال (registers) اللي هستخدمها وفي الآخر هعمل (PoP) ليهم بس بالعكس.

Push 4
Push 5
PoP 5
PoP 4

لو عملت (Push) ومفيش في الآخر (PoP) مش هيطلع قيم ال (PC) الطلوبة.

لو لازم كل (Push) في البداية يقابلها نفس ال (PoP)

---

* ال (instruction) يتم تقريبة بال (crystal oscillator)

ال machine cycle : هي (cycle) يتم تنفيذ الأمر فيها

لو ال (microcontroller) يحتوي على مجموعة من ال (operations) كل (operation) تحتاج كام (cycle)

ال (machine cycle) لل (atmel) عبارة عن (12 clocks)

→ Find Period of machine cycle for 11.0592 MHz Crystal Frequency.

$$\frac{11.0592}{12} = 921.6\ KHz \quad \& \quad Machine\ cycle = \frac{1}{921.6KHz} = 1.085\ \mu s$$

Addressing modes

1) Immediate
2) register.
3) Direct
4) register indirect
5) Indexed.

↳ Accessing memories.

[1] Immediate

↳ src. operand is constant. MOV A, #25H

↳ load info. into any registers, including 16-bit DPTR register.

[2] Register

↳ hold data to be manipulated (MOV A, R0)

↳ src. & dst. registers must match in size

ex MOV DPTR, A (XX) & MOV DPTR, #25F5H

↳ MOV R4, R7 (XX)

[3] Direct addressing mode

↳ used to access RAM locations 30-7FH

ex: MOV A, 4 & MOV A, R4 (no # sign here)

* SFR (special function register)

↳ Can be accessed by their names or their addresses

↳ have addresses between 80H and FFH.

MOV 0E0H, #55H
MOV A, #55H
MOV 0F0H, R0
MOV B, R0

EX write code to send 55H to ports P1 & P2 using.

a) their names

```
Mov A, #55H
Mov P1, A
Mov P2, A
```

b) their addresses

```
Mov A, #55H
Mov 80H, A
Mov 0A0H, A
```

---

*stack and direct addressing mode

↳only direct addressing mode is allowed for pushing or popping the stack.

↳Push A (invalid) ⇒ it should be Push 0E0H

EX show the code to Push R5 and A onto the stack and then pop them back into R2 & B, (B=A, R2=R5)

sol

```
Push 05        ; Push R5 to stack
Push 0E0H      ; Push register A to "
Pop  0F0H      ; Pop top of stack to B
                 now B = A
Pop  02        ; Pop top of stack into R2
                 now R2 = R5
```

4 register indirect addressing mode :-

↳ register is used as a pointer to data (only R0, R1)

MOV A, @R0 → move contents of RAM whose address is held by R0 into A

MOV @R1, B

EX write a program to copy value 55H into RAM memory locations 40H to 41H using :

a) direct addressing mode

```
MOV A, #55H
MOV 40H, A
MOV 41H, A
```

b) register indirect without loop

```
MOV A, #55H
MOV R0, #40H
MOV @R0, A
INC R0
MOV @R0, A
```

c) register indirect with loop

```
    MOV A, #55H
    MOV R0, #40H
    MOV R2, #02
AGAIN: MOV @R0, A
    INC R0
    DJNZ R2, AGAIN
```

↳ advantage of register indirect that it makes accessing data dynamic rather than static as in direct addressing mode.

EX write Program to clear 16 RAM locations starting at RAM address 60H

s:1

```
          clear A          ; A=0
          MOV R1, #60H     ; R1=60H Pointer
          MOV R7, #16      ; Load counter R7=16
AGAIN :   MOV @R1, A       ; clear RAM R1 points to
          INC R1           :
          DJNZ R7, AGAIN   ; loop until counter=zero
```

EX: write Program to Copy a block of 10 bytes of data from 35H to 60H

solution

```
          MOV R0, #35H
          MOV R1, #60H
          MOV R2, 10
Back :    MOV A, @R0
          MOV @R1, A
          INC R0
          INC R1
          DJNZ R2, Back
```

Indexed addressing mode

↳ widely used in accessing data elements of look-up table entries located in Program ROM.

instruction used ⇒ MOVC, @A+DPTR
↳ code ↳ 16-bit register

~~↳ contents of~~

Note: look for ex (5-6 & 5-8) in slides Page 175, 176

Notes

→ In many apps, size of Program Code doesn't leave any room to share 64K-byte Code space with data.
↳ 8051 has another 64K-bytes of space set aside for data storage.

↳ 8051 has 128K-bytes of memory space
↳ 64K bytes of Code + 64 bytes of data

→ data space between Code & data Cannot be shared.

→ In many apps. we use RAM locations 30-7FH as scratch Pad.

*write Program to toggle P1 → 200 times. use RAM location 32 to hold you Counter value instead of R0-R7 register

```
        MOV P1, #55H
        MOV 32H, #200
LOP1:   CPL  P1
        ACALL DELAY
        DJNZ 32H, LOP1
```

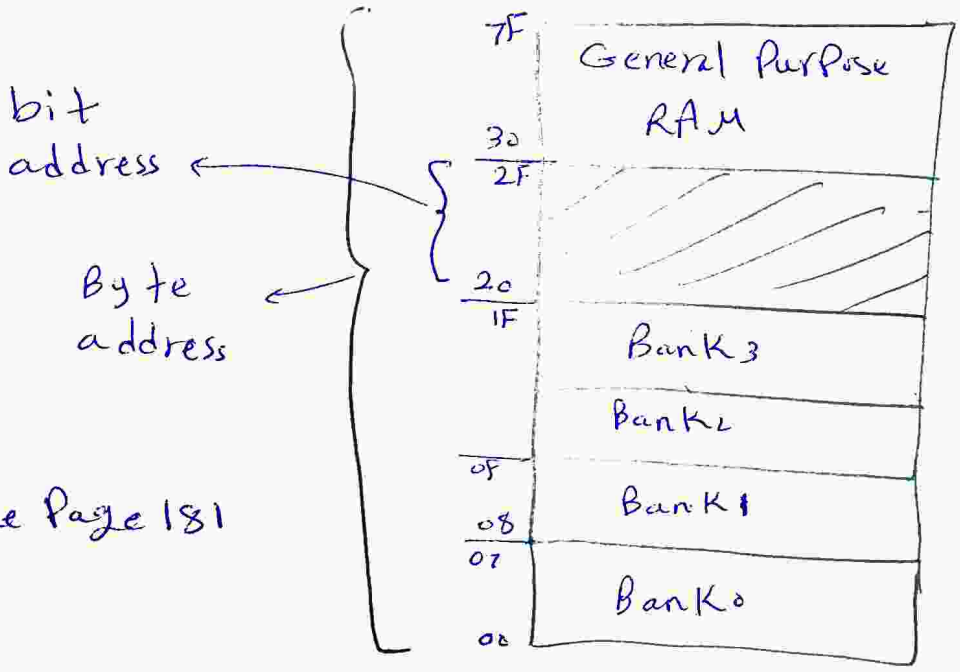
Bit Addresses

⇒ Many microProcessors allow Program to access registers and I/o ports in byte size.

↳ 8051 is single-bit operation ⇒ allow programmer to ~~do operations~~ set, clear, move, complement bits of port or memory or register.

ROM → only not bit-addressable.

* Internal RAM locations 20-2FH are (byte & bit) addressable
  ↳ 00-7FH



look for example Page 181

↳ 00-7FH Can be accessed in byte size using direct & ~~ind~~ register indirect addressing mode.

→ Instructions here are single bit like:-

JB bit, Target — if bit = 1 → Jump to target

JBC bit, target — " " " → " " " ; then clear bit.

Ports $P_0 - P_3$ ⇒ bit addressable

*when we access Port in single bit

SETB X.Y

X → Portnumber (P_0, P_1, P_2, P_3)

Y → desired bit no. from 0 to 7

registers ⇒ A, B, PSW, IP, IE, ACC, SCON & TCON

↳ are bit addressable.

EX write Program to save accumulator in R7 of bank 2.

```
CLR  PSW.3
SETB PSW.4
MOV  R7, A
```

look for examples at Page 188 & 189

*BIT Directive used to assign bit-addressable I/O and RAM locations.

EX switch is Connected to Pin P1.7 and LED to Pin P2.0 write a program to get status of switch and send it to the LED

```
LED     BIT  P1.7
SW      BIT  P2.0
HERE:   MOV  C, SW
        MOV  LED, C
        SJMP HERE
```

EQU in bit addresses : assign addresses

↳ Defined by names (like P1.7)

↳ " " addresses (~~97H~~ like 97H)

→ Example at Page 192.

*8052 has another 128 bytes of on-chip RAM with addresses 80-FFH → called upper memory.

MOV @R0, A & MOV @R1, A

→ example Page 194

Arithmetic & Logic instructions and Programs

*Addition of unsigned numbers:- Add A, src

↳ Destination is always in register A.

↳ src. operand can be register, ~~inder~~ immediate data or in memory.

↳ memory-to-memory arithmetic operations not allowed in 8051.

*show How flag register is affected by this:

```
MOV A, #0F5H  ; A=F5
ADD A, #0BH   ; A=F5+0B =00
```

	Sol	
F5H	1111	0101
0BH	0000	1011
100H	0000	0000

CY = 1

PF = 1 (no. of 1s zero)

AC = 1 → carry from D3 to D4

Addition of individual bytes

→ Assume that RAM locations 40-44H have these values. write Program to find sum of values. at the end of Program, register A should contain low byte & R7 the high byte.

40 = (70) , 41 = (EB) , 42 = (C5) , 43 = (5B) , 44 = (30)

Sol

```
        MOV R0, #40H   ; load Pointer
        MOV R2, #5     ; "  Counter
        CLR A          ; A = 0
        MOV R7, A      ; clear R7
AGAIN:  ADD A, @R0     ; add byte R0 to
        JNC NEXT       ; if CY = 0 don't add carry
        INC R7         ; keep track of carry
NEXT:   INC R0         ; increment pointer
        DJNZ R2, AGAIN ; repeat until R2 = 0
```

Note

↳ when adding two 16-bit data operands, Propagation of carry from lower byte to higher byte is concerned.

Example at 198.

BCD Binary coded decimal

↳ binary representation of digits 0 to 9

BCD → Packed BCD
BCD → unPacked.

1 unPacked BCD

↳ lower 4 bits of number represent BCD number + rest of bits are zero

EX 00001001 → unPacked BCD for 9.

2 Packed BCD

↳ single byte has two number (one in lower 4-bits and one in the upper 4 bits)

EX 0101 1001 → is Packed BCD for 59H

↳ Adding two BCD numbers must give BCD result.

EX

MOV A, #17H

ADD A, #28H

⇒ result = 3FH → it should be 17+28=45

↳ correction 3F+06 = 45H

DA: decimal adjust for addition

⇒ After ADD or ADDC instruction

1) if lower (4 bits) > 9 or AC = 1 add (0110) to lower 4-bits.

2) if upper nibble (4-bits) > 9 or CY = 1 add (0110) to upper 4 bits.

EX

29	0010	1001
18	0001	1000
41	0100	0001
6		0110
47	0100	0111

AC = 1

Example at Page 203

Add Program to find sum of numbers. result must be in BCD.

40 = (71), 41 = (11), 42 = (65), 43 = (59), 44 = (37)

5:1

```
        MOV  R0, #40H        |        DA   A      → adjust BCD
        MOV  R2, #5          |        JNC  NEXT
        CLR  A               |        INC  R7
        MOV  R7, A           |  NEXT : INC  R0
AGAIN : ADD  A, @R0          |        DJNZ R2, AGAIN
```

In microprocessor : Subtraction

SUB

SUBB → sub with borrow

In 8051 ⇒ only SUBB

SUBB A, src ⇒ A = A − src − CY

↳ to get SUB from SUBB ⇒ Put CY = 0

~~In this case~~ SUBB when CY = 0

1) take 2's complement of src. operand.
2) Add it to A
3) invert the Carry.

```
   AC          0100 1100        0100 1100
 - 6E          0110 1110  2's → 1001 0010   +
-----          ---------        ---------
  -22          CY=1  ←          0110 1110
```

* SUBB when CY = 1

→ used for multi-byte numbers & will take care of borrow of lower operand.

2762H − 1296H = 14CCH

→ 8051 supports byte by byte multiplication

MUL AB ; A * B

ex: Page 207

Division

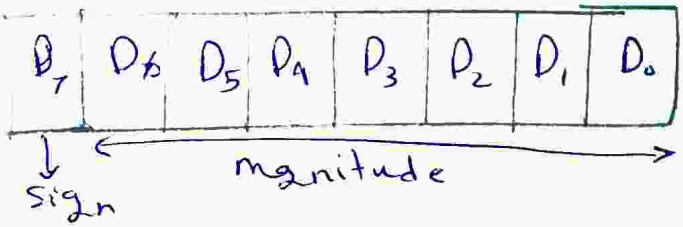
DIV AB ; A/B $\frac{95}{10} \Rightarrow$ 09 → quotient, 05 → reminder

→ look for example at page 209

signed operands

+ve numbers ⇒ 0 to +127

-ve " ('2's Complement)



overflow (happens if result of operation on signed numbers is too large for register.

ex

$$\begin{array}{r} 96 \\ +\ 70 \\ \hline 166 \end{array} \qquad \begin{array}{r} 0110\ 0000 \\ 0100\ 0110 \\ \hline 1010\ 0110 \end{array} \Rightarrow OV = 1$$

* In 8-bit signed numbers operations, OV set to 1 if either occur:

1) there is carry from D_6 to D_7, but no carry out of D_7 (CY = 0)

2) there is carry from D_7 out (CY = 1), but no carry from D_6 to D_7.

notes

1) in signed number addition, we must monitor status of CY. use JNC or JC instructions.

2) in signed no. add, OV flag must be monitored.

JB PSW.2 or JNB PSW.2

=> 2's complement of no. => CPL A
ADD A, #1

AND instruction (ANL)

→ destination is Accumulator

→ src can be register, memory or immediate

X	X	S
0	0	0
0	1	0
1	0	0
1	1	1

EX

```
MOV  A, #35H
ANL  A, #0FH
```

OR operation (ORL)

X	Y	S
0	0	0
0	1	1
1	0	1
1	1	1

EX

```
MOV A, #04H
ORL A, #68
```

```
0000 0100
0110 1000
-----------
0110 1100
```

XOR operation (XRL)

X	X	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Compare instruction (CJNE dst, src, rel.addr.)

↳ Compare two operands, Jump if they are not equal.

↳ dst. is Accumulator or one of Rn registers.

↳ changes CY to see if dst. is larger or smaller.

EX

```
CJNE R5, #80, NOT_EQual
- - - -
NOT_Equal:
```

note in CJNE instruction, any Rn registers can be compared with immedate value.

↳ Comparing is Subtraction except that operands remain unchanged.

EX: at Page 224

Rotate instruction RR · A rotate right

EX 0011 0110 ⇒ 0001 1011

* RL A rotate left

EX 0011 0110 ⇒ 0110 1100

*Rotating through Carry ~~RC RRCA~~

1 RRC A

CY=0 ⇒ ~~0010~~ 0010 0110 → 0001 0011

CY=1 ⇒ 0001 0011 → 0000 1001

2 RLC A (same but left)

~~CY A~~

Serializing data

↳ sending byte of data one bit at a time through a single pin of micro controller.

using serial Port: to transfer one bit at a time & control sequence of data and spaces between them.

examples at Page 230, 231 | table of Carry operations at Page 232, 233

Swap

↳ It swaps the lower and higher nibles بعكس بين

before: D7-D4 D3-D0 ⇒ D3-D0 D7-D4

EX
```
MOV A, #72H    ; A = 72H
SWAP A         ; A = 27H
```

notes

* DS5000T microcontrollers have real-time clock (RTC)
* To Convert ASCII to Packed BCD

Key	ASCII	unPacked BCD
4	34	0000 0100 ⇒ 01000111 or 47H
7	37	0000 0111

↝ example at Page 239 & 240

note

To ensure the integrity of ROM contents, every system must perform checksum calculation

Checksum

↳ detects any corruption of contents of ROM.

↳ used checksum byte

To calculate checksum byte of series of bytes

↳ Add bytes together & drop carries.

↳ take 2's complement to total sum, it becomes last byte of seriers.

To perform checksum operation, add all bytes including checksum byte.

↳ result must be zero.

↳ if not zero, one or more bytes of data have been changed.

(EX) for these 25H, 62H, 3FH, 52H

a) find checksum byte

25 H
62 H
3F H
52 H

118

drop carry you get 18

2's complement " " E8H

b) perform checksum to ensure integrity

25H
62H
3FH
52H
E8H

200

drop carry

00

8051 Programming in C

Why Program in C

1) less time consuming but larger hex file size
2) easier than assembly.
3) easy to modify & update.
4) Can use code available in function libraries.

Data types

1) unsigned char :: 8-bit at range 0-255

↳ ~~widely used~~

↳ C Compilers use signed char as default if we don't put the keyword unsigned

Examples at Page 249 & 250.

2] signed char (8-bit data type)

↳ use MSB D7 to represent - or +

↳ Give values from -128 to +127.

* Example at Page 251

*single Bit (sbit)

↳ allow access to the single bits of SFR registers.

example at Page 253

*Bit & sfr

↳ bit datatype allow access to single bits of bit-addressable memory spaces 20-2FH.

↳ to access byte-size SFR registers, use sfr datatype.

Time delay to create it:

1) using 8051 timer.

2) using simple for loop

Ex write 8051 program to toggle bits of P1 continuously forever with some delay.

Sol

```
#include <reg51.h>
void main (void)
{
unsigned int x;
for (;;)
{
P1 = 0x55;
for(x=0;x<40000;x++);
P1 = 0xAA;
for (x=0;x<40000;x++);
}
}
```

---

toggle ~~P1~~ bits of P1 Port with delay 250 ms

```
#include <reg51.h>
void MsDelay (unsigned char)
void main (void)
{
while (1)
{
P1 = 0x55;
MsDelay (250);
P1 = 0xAA;
MsDelay (250);
}}
Void MsDelay (unsigned int itime)
{unsigned int i,j;
for (i=0;i<itime;i++);
for (j=0;j<1275;j+4);
}
```

See examples from 258 to 267

operators

AND (&&) , OR (||) , NOT (!)

shift right (>>) , inverter (~) , EX-OR (^)

See examples from 269 to 278

* 8501 C compiler allocates RAM locations
1) Bank 0 (0-7)
*2) Individual variables.
3) Array elements
4) stack

examples 279, 280

* extra 128 bytes of RAM helps 8051/52 C-compiler to manage its registers & resources more effectively.

* To make C-compiler use code space instead of RAM space, Put Keyword "code" in front of variable declaration as in ex: Page 283.

at Page 284 & 285 → discussing programs (may be important)

Interrupts chapter

Interrupt: external or internal event that interrupt microcontroller to inform it the a device needs its service.

*single microcontroller can serve devices by:

*Interrupts

↳whenever any device need its service, it notifies the microcontroller by send it interrupt signal

↳them MC stop whatever it does, and serve the device

*Polling

↳MC continuously monitors the status of a given device

↳when conditions met, it performs the device.

↳then, it moves on to monitor next device until serve all devices.

Advantage:

↳MC can serve many devices

Disadvantage

↳Can't serve all devices at the same time

Advantage

↳can serve all devices if cnditions are met

Disadvantage

↳not efficient, cause it wastes time to MCs that not need service.

*Interrupt service routine (ISR)

↳for every interrupt, there is fixed location in memory that hold address of its ISR

↳program associated with interrupt.

* steps in Executing interrupt :- MC go in these steps:

1) finishes instruction it is executing and saves address of next instruction on stack.

2) saves current status of all interrupt internally (not in stack)

3) It jump to a fixed location in memory, called interrupt vector table, that holds address of ISR.

4) MC gets address of ISR and jump to it.
   ↳ starts to execute ISR until reaching last inst. of subroutine (RETI : return from interrupt)

5) upon executing RETI inst., Mc returns to the place where it was interrupted

---

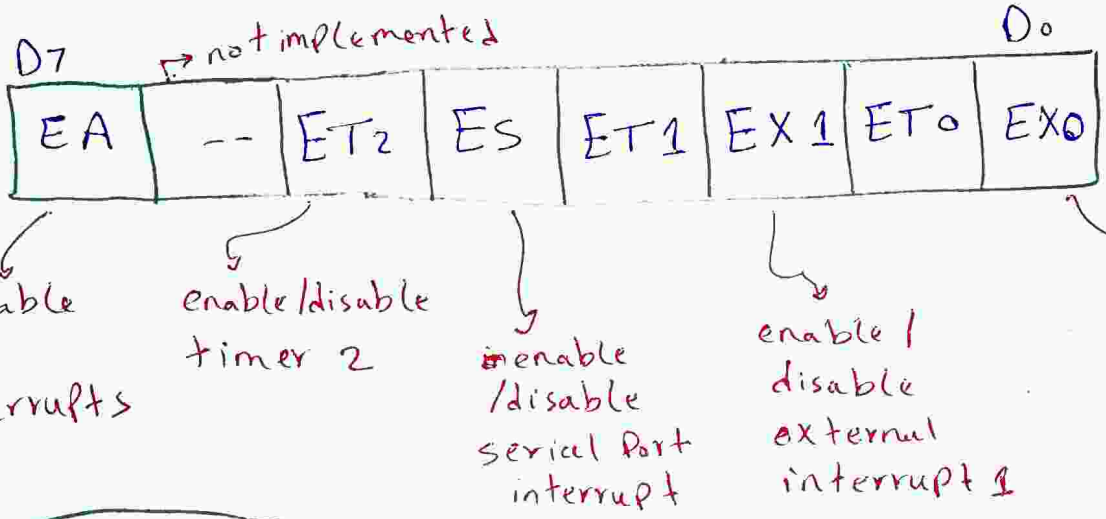
Six interrupts at 8051

1) reset & power-up reset
2) Two for timers
3) Two for Hw external interrupts (P3.2 & P3.3) — INT0 , INT1 ↑
4) serial communication has single interrupt that belong to receive & transfer.

P. 127 Enable & Disable interrupt

* upon reset, all interrupts are disabled.
* interrupts must be enabled by SW so that microcontroller could respond to them.

IE: (interrupt enable) → enable & disable interrupt



Steps of enabling an interrupt

1) Bit D7 must be set to high

2) Value of EA
   - ↳ if EA = 1, interrupts are enabled & will be responded to if bits in IE are high.
   - ↳ if EA = 0, interrupts are disabled, ~~not~~ not responded.

Example at Page 430.

Timer interrupts

↳ Timer flag (TF) is raised when timer rolls over

Problem in polling :-

↳ we have to wait until TF is raised

so, micro controller is tied down while waiting for raising of TF, and can't do anything else.

How Interrupts solve this problem?

↳ if timer interrupt in IE register enabled, whenever time rolls over (TF is raised) & MC is interrupted in whatever it does, and Jumps to interrupt vector table to serve ISR.

↳ So that MC can do any thing until it is notified that TF is raised.

see examples from 432 to 436 at slides

External HW Interrupts

P3.2 & P3.3 ⇒ INT0 & INT1

* activation levels for external HW interrupts
  - ↳ level triggered
  - ↳ Edge "

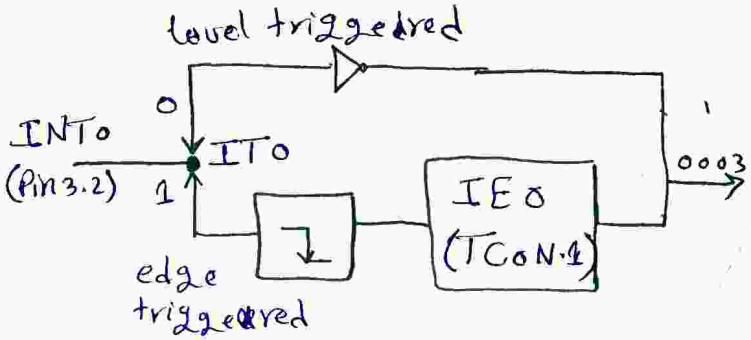
Level triggered mode

↳ INT0 (INT1 normally high.

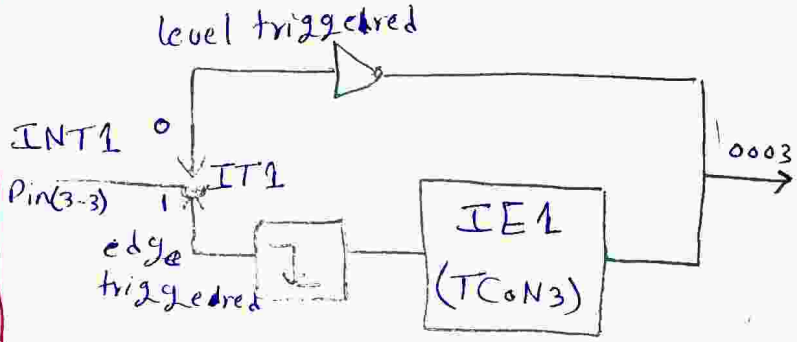
↳ if low-level signal is applied to them, it triggers interrupt.

↳ then Mc stops whatever it's doing & jump to interrupt vector table to serve this interrupt.

↳ low-level signal at INT pin must be removed before executing last instruction of ISR (RETI); otherwise another interrupt will be generated



Activation of INT0



Activation of INT1

---

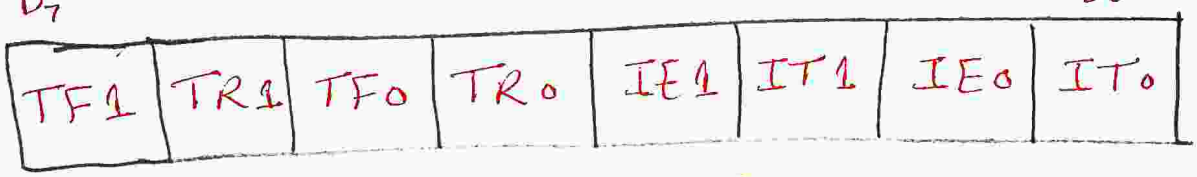
see examples at Page 440

Note Pins P3.2 & P3.3 are used for normal I/O unless INT0 & INT1 bits in IE are enabled

---

* To make INT0, INT1 edge-triggered interrupts
↳ we must program bits of TCON register.
↳ IT0, IT1 are bits D0 & D2 of TCON register.
↳ They are referred to as TCON.0 & TCON.2
since TCON is bit addressable.

TCON



* TF1 (TCON.7) → timer 1 overflag.
  - ↳ set by HW when timer/counter 1 overflows.
  - ↳ cleared by HW.

TR1 (TCON.6) timer 1 run control bit.
- ↳ set/cleared by SW turn timer/counter 1 on/off.

TF0 (TCON.5) → timer 0 over flag.

TR0 (TCON.4) → timer 0 run control bit

IE1 (TCON.3) → External interrupt 1 edge flag.
- ↳ set by CPU when external interrupt edge detected.
- ↳ cleared by CPU when interrupt is processed.

IT1 (TCON.1) → interrupt 1 type control bit
- ↳ set/cleared by SW to specify failing edge/ low-level triggered external interrupt.

see example at Page 446

Notes

- At edge-triggered interrupts:-
  - External source must be held high for at least one machine cycle & then held low for at least " " " .
  - Failing edge of INT0 & INT1 are latched by 8051 & are held by TCON.1 and TCON.3

Ex what is the difference between RET and RETI instructions? How we can't use " instead of RETI?

- Both perform same action of popping of two bytes of stack into PC, and marking 8051 return to left off

But, RETI perform additional task of clearing the interrupt-in-service flag

If you use RET instead of RETI in last instruction of ISR → you blocks any new interrupt on that pin after first interrupt, cause RET don't clear interrupt in service flag.

Serial Communication Interrupt

TI (transfer interrupt)

↳ raised when last bit of framed data (stop bit) is transfered

↳ indicating that SBUF register read to transfer next byte.

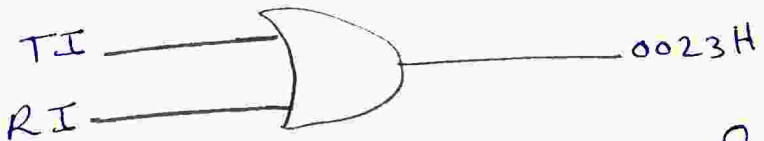
RI (received interrupt)

↳ raised when entire frame of data including stop bit is received.

In 8051 there is one interrupt for serial communication.

↳ used to send & receive data.

if IE.4 enabled, when RI or TI is raised, the 8051 get interrupted and jump to memory location 0023H to execute ISR.



* Serial interrupt is mainly used for receiving data & is never used for sending.

See examples from Page 453 to 458

Interrupt Priority

↳ when 8051 is powered up, the priorities are assigned according to:

Highest to lowest	Priority
External interrupt 0	INT0
timer " 0	TF0
External " 1	INT1
timer " 1	TF1
Serial Communication	RI + TI

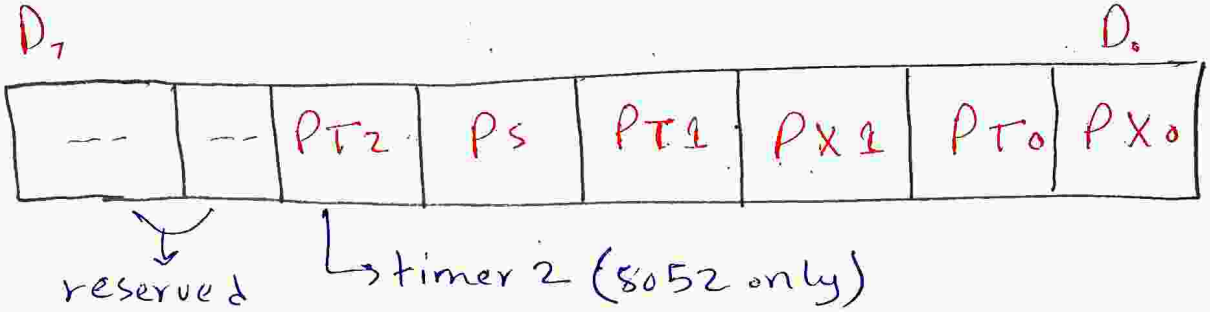
Interrupt priority upon reset

in Page 461 example ask what if more than one interrupt activated at same time → answer will be this Priority.

IP (interrupt priority) used to give higher Priority to interrupts, we make bit in IP register → high.

→ if you have two or more interrupt bits in IP register high ⇒ you will go to the priority table.

Interrupt Priority register



See examples at Page 464 & 465 (important)

Note in 8051 : low priority interrupt can be interrupted ~~by~~ by higher " " not by another low - " " .

Programming in C

→ compiler C

* ~~They~~ can assign unique number of 8051 interrupts

* It can assign register bank to an ISR

interrupt name	number
INT 0	0
TF 0	1
INT 1	2
TF 1	3
RI + RI	4
TF 2	5

Examples at Page 469 & 470